Generic On-line/Off-line Aggregate Signatures*

Chunhui Wu[1], Yuqing Xing[1], Xiaofeng Chen[1], Dongyang Long[1],
Hyunrok Lee[2], and Kwangjo Kim[2]
[1]School of Information Science and Technology,
Sun Yat-sen University, Guangzhou 510275, P.R.China
[2]International Research center for Information Security (IRIS)
Information and Communications University(ICU), Taejon 305-714, KOREA
Email: isschxf@mail.sysu.edu.cn; {tank,kkj}@icu.ac.kr

Abstract

Aggregate signatures play an important role in many real-world applications involve signatures on many different messages generated by many different users. However, the existing aggregate signature schemes are unsuitable for the environments where the computing and storage resources are limited since it always involves in inefficient modular exponentiation or pairing operations in aggregate signing or verification algorithms. In this paper we introduce the concept of on-line/off-line aggregate signatures in order to improve the performance of aggregate signatures. We also use an efficient double-trapdoor chameleon hash function to present a concrete construction based on the so-called "hash-sign-switch" paradigm. Moreover, we give an application of our scheme in stock markets, where the stockbroker can compress and batch verify the stockholders' signatures in time due to the on-line/off-line property.

1. Introduction

An aggregate signature scheme is a digital signature that support aggregation: Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single short signature token. This single signature token (and some potentially additional information) will convince the verifier that the n users indeed signed the n original messages.

Aggregate signatures are useful in many real-world applications. For example, certificate chains in a hierarchical PKI of depth n consist of n signatures by n different CAs on n different public keys; by using an aggregate signature scheme, this chain can be compressed down to a single aggregate certificate. Another application is secure routing. In secure BGP [2], each router successively signs its segment of a path in the network, and forwards the collection of signatures associated with the path to the next router; forwarding these signatures entails a high transmission overhead that could be reduced by using aggregate signatures. Besides the property of compactness, aggregate signatures have other advantages. For example, a signature that has been aggregated cannot be separated. This is useful in scenarios such as database outsourcing [8] and dynamic content distribution [9] where one may want to prevent a malicious party from removing a signature from a collection of signatures without being detected.

There are several works on aggregate signatures. Boneh *et al* [1] proposed the first *flexible* aggregate signature scheme using bilinear maps. Given n individual signatures $\sigma_1, \sigma_2, \cdots, \sigma_n$, anyone can aggregate them in any order into an aggregate signature σ. Lysyanskaya *et al* [7] proposed a *sequential* aggregate signature scheme using a weaker assumption, namely, certified trapdoor permutations, but the aggregation must performed in order, *i.e.*, the n-th signer must aggregate its own signature into the aggregate signature formed by the first $n-1$ signers.

In PKI settings, the total verification information $\mathcal{V}$ must contain individual signer public keys, beyond the information-theoretic minimum verification information $\mathcal{D}$ of describing who signed what. Lately, Gentry and Ramzan [5] proposed an identity-based aggregate signature scheme in order to minimize the total verification information and to reduce the communication complexity, *i.e.*, eliminating the delivering of public keys, trying to get close to the information-theoretic lower bound. The scheme in [5] is also very efficient, with 3 pairing operations for verification.

The notion of on-line/off-line signatures was introduced by Even, Goldreich and Micali [4]. It performs the signature generating procedure in two phases. The first phase is per-

*Supported by National Natural Science Foundation of China (No. 60503006) and NSFC-KOSEF Joint Research Project (No. 60611140543).

formed *off-line* (without knowing the signed message) and the second phase is performed *on-line* (after knowing the signed message). The results of the pre-computation in the off-line phase are saved and then used in the on-line phase when a message must be signed. Shamir and Tauman [10] used the so-called "chameleon hash functions" to develop a new paradigm, named "hash-sign-switch", for designing much more efficient on-line/off-line signature schemes. Recently, Chen *et al* proposed an efficient double-trapdoor chameleon hash function [3] to reduce the computation and storage overload in on-line/off-line signatures, by reusing the same chameleon hash value without exposuring the *long-term* trapdoor key.

In this paper, we propose an on-line/off-line aggregate signature scheme. To the best of our knowledge, it is the first on-line/off-line aggregate signature scheme. Our goal is to improve the performance of aggregate signatures. It is suitable for applications in the environments with limited computation resources. For example, in secure routing protocol SBGP, though the routers have limited computation resources, they can do some pre-computation of signing and verifying off-line. It is also suitable for aggregate applications with "bursty traffic". In the authorizing stage of a voting scheme, the administrator signs for valid voters in a short period; the administrator can use on-line/off-line aggregate signatures to do some pre-computation off-line and aggregate the signatures. Our scheme can also be used in the stock markets. For more details, please refer to section 4.3.

The rest of the paper is organized as follows. After providing some preliminaries in Section 2, we propose a formal definition and security model for on-line/off-line aggregate signatures in section 3. We construct on-line/off-line aggregate signature schemes based on the "hash-sign-switch" paradigm in section 4. We give the security and efficiency analysis in section 5. Finally, we conclude in section 6.

2. Preliminaries

In this section, we introduce the notion of chameleon hash family and the "hash-sign-switch" paradigm [10].

Definition 1. *A chameleon hash family consists of a pair* $(\mathcal{I}, \mathcal{H})$*:*

- $\mathcal{I}$ *is a probabilistic polynomial-time key generation algorithm that on input* 1^k*, outputs a hash/trapdoor pair* (HK, TK) *such that the sizes of* HK, TK *are polynomially related to k.*
- $\mathcal{H}$ *is a family of randomized hash functions. Every hash function in* $\mathcal{H}$ *is associated with a hash key* HK*, and is applied to a message from a space* $\mathcal{M}$ *and a random element from a finite space* $\mathcal{R}$*. The output of the hash function* H_{HK} *does not depend on* TK*.*

A chameleon hash family $(\mathcal{I}, \mathcal{H})$ should have the properties of efficiency, collision resistance, and trapdoor collisions.

Definition 2. *Shamir and Tauman introduced the following "hash-sign-switch" paradigm to get a generic on-line/off-line signature scheme.*

- **System Parameters Generation:** *Let* $(\mathcal{I}, \mathcal{H})$ *be any trapdoor hash family and* $(\mathcal{G}, \mathcal{S}, \mathcal{V})$ *be any provably secure signature scheme. The system parameters are* $SP = \{(\mathcal{I}, \mathcal{H}), (\mathcal{G}, \mathcal{S}, \mathcal{V})\}$.
- **Key Generation Algorithm:**
  - *On input* 1^k*, run the key generation algorithm of the original signature scheme* $\mathcal{G}$ *to obtain a private/public key pair* (SK, VK).
  - *On input* 1^k*, run the key generation algorithm of the trapdoor hash family* $(\mathcal{I}, \mathcal{H})$ *to obtain a trapdoor/hash key pair* (TK, HK).

  The signing key is (SK, TK) *and the verification key is* (VK, HK).
- **The Signing Algorithm:**
  1. *Off-line phase*
     - *Choose at random* $(m_i, r_i) \in_R \mathcal{M} \times \mathcal{R}$*, and compute the chameleon hash value* $h_i = H_{HK}(m_i, r_i)$.
     - *Run the signing algorithm* $\mathcal{S}$ *with the signing key* SK *to sign the message* h_i*. Let the output be* $\sigma_i = \mathcal{S}_{SK}(h_i)$.
     - *Store the pair* (m_i, r_i)*, and the signature* σ_i.
  2. *On-line phase*
     - *For a given message* m*, retrieve from the memory a random pair* (m_i, r_i) *and the signature* σ_i.
     - *Compute* $r \in \mathcal{R}$ *such that* $H_{HK}(m, r) = H_{HK}(m_i, r_i)$.
     - *Send* (r, σ_i) *as the signature of message* m.
- **The Verification Algorithm:**
  - *Compute* $h_i = H_{HK}(m, r)$.
  - *Verify that* σ_i *is indeed a signature of the hash value* h_i *with respect to the public key* VK.

3 On-line/Off-line Aggregate Signatures

In this section, we present the formal definition and security model of on-line/off-line *flexible* aggregate signature.

3.1. Definitions

An on-line/off-line aggregate signature scheme is composed of six algorithms: system parameter generation (Sys-Par-Gen), on-line/off-line aggregate key generation (On/Off-Agg-Key-Gen), off-line aggregate sign (Off-Agg-Sig), off-line aggregate verify (Off-Agg-Ver), on-line aggregate sign (On-Agg-Sign), and on-line aggregate verify (On-Agg-Ver).

- Sys-Par-Gen (Done Once): takes 1^λ as input and output system parameters, such as chameleon hash function and bilinear group.
- On/Off-Agg-Key-Gen (Done Once): takes 1^k and system parameters as input, the common outputs include on-line/off-line verification key (VK_j, HK_j) of each player u_j; the private outputs include individual on-line/off-line signing key (SK_j, TK_j).
- Off-Agg-Sign (Done per Aggregation): takes the aggregate player set $\mathcal{U}$, and private keys as input; the common outputs include the single aggregate signature token σ' for the set $\mathcal{U}$; the private outputs include pre-computed signature information.
- Off-Agg-Ver (Done per Aggregation): takes the aggregate player set $\mathcal{U}$, chameleon hash value H'_j of player u_j, public keys, and the aggregate signature token σ' as input; output 1 if the aggregate signature token is valid.
- On-Agg-Sign (Done per Aggregation): takes the aggregate player set $\mathcal{U}$, messages m_j to be signed, trapdoor keys, and pre-computed signature information stored off-line as input; output the on-line/off-line aggregate signature Σ.
- On-Agg-Ver (Done per Aggregation): takes the aggregate player set $\mathcal{U}$, chameleon hash value H'_j, hash keys, and the aggregate signature Σ as input; output 1 if the aggregate signature is valid, *i.e.*, chameleon commitments are valid.

3.2. Security Model

Informally, the security of on-line/off-line aggregate signature schemes is equivalent to the nonexistence of an adversary capable, within the confines of a certain game, of existentially forging an on-line/off-line aggregate signature. Existential forgery here means that the adversary attempts to forge an on-line/off-line aggregate signature, on messages of his choice, by some set of players.

Similar to [1], we also consider the "aggregate chosen-key security model" for on-line/off-line aggregate signature schemes. In this model, the adversary is given a single public key and a single hash key and his goal is the existential forgery of an on-line/off-line aggregate signature. We give the adversary power to choose all public keys except the challenge public key, and to choose all hash keys except the challenge hash key. The adversary is also given access to a signing oracle and to a chameleon collision oracle on the challenge keys. His advantage, $\mathsf{AdvAggSig}_{\mathsf{A}}^{\mathsf{On/Off}}$, is defined to be his probability of success in the following game.

- **Setup.** The aggregate forger $\mathcal{A}$ is provided with a public key PK_1 and a hash key HK_1, generated at random.
- **Queries.** Proceeding adaptively, $\mathcal{A}$ requires chameleon hash collisions with HK_1 and the corresponding signatures with PK_1 on messages of his choice.
- **Response.** Finally, $\mathcal{A}$ outputs $n-1$ additional public keys $PK_2, PK_3, \cdots, PK_n$; and $n-1$ additional hash keys $HK_2, HK_3, \cdots, HK_n$. Here n is at most m, a game parameter. These keys, along with the initial keys PK_1 and HK_1, will be included in $\mathcal{A}$'s forged on-line/off-line aggregate signature. $\mathcal{A}$ also outputs messages $m_1, m_2, \cdots, m_n$; and finally, an on-line/off-line aggregate signature Σ by the n users, each on his corresponding message.

  The forger wins if the on-line/off-line aggregate signature Σ is a valid aggregate on messages $m_1, m_2, \cdots, m_n$ under keys $PK_1, PK_2, \cdots, PK_n$ and $HK_1, HK_2, \cdots, HK_n$; and Σ is nontrivial, *i.e.*, $\mathcal{A}$ did not request a chameleon collision on m_1 under HK_1 and the corresponding signature under PK_1. The probability is over the coin tosses of the key-generation algorithm and of $\mathcal{A}$.

Definition 3. *An aggregate forger $\mathcal{A}(t, q_C, q_S, m, \epsilon)$-breaks a m-player on-line/off-line aggregate signature scheme in the on-line/off-line aggregate chosen-key model if: $\mathcal{A}$ runs in time at most t; makes at most q_C queries to the chameleon collision oracle and at most q_S queries to the signing oracle; $AdvAggSig_A^{On/Off}$ is at least ϵ; and the forged on-line/off-line aggregate signature is by at most m players. An on-line/off-line aggregate signature scheme is $(t, q_C, q_S, m, \epsilon)$-secure against existential forgery in the on-line/off-line aggregate chosen-key model if no forger $(t, q_C, q_S, m, \epsilon)$-breaks it.*

4 Efficient On-line/Off-line Aggregate Signature Schemes

In this section, we first show how to convert Boneh *et al*'s aggregate signature [1] into an on-line/off-line aggregate signature. We then give a new efficient construction based on the "hash-sign-switch" paradigm.

4.1 On-line/Off-line Aggregate Signature Based on Boneh's Signature

We can convert Boneh *et al*'s aggregate signature [1] to an on-line/off-line aggregate signature based on the "hash-sign-switch" paradigm, using Peterson's commitment. Denote the aggregate player set by $\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$, where $n < m$, the maximum number of players.

- Sys-Par-Gen (Done Once):

  Let l be a prime power, and $E(F_l)$ be an elliptic curve over finite field F_l. Let $\#E(F_l)$ be the number of points of $E(F_l)$, and P be a point of $E(F_l)$ with prime order q where $q|\#E(F_l)$. Denote G the subgroup generated by P. Assume that G is a Gap Diffie-Hellman group and there exist a bilinear map e: $G \times G \rightarrow G_T$. Given a hash key $HK = Y$, where $Y = yP$ and $y \in_R Z_q^*$, the chameleon hash function $H_{HK} : Z_q \times Z_q \rightarrow G$ is defined as follows:

  $$H_{HK}(m, r) = mY + rP$$

  The system parameters are $SP = \{E, l, q, P, G, H_{HK}\}$.

- On/Off-Agg-Key-Gen (Done Once):

  1. Each player u_j runs the key generation algorithm of the signature scheme, obtains the private/public key pair $(x_j, V_j = x_jP)$.
  2. Each player u_j runs the key generation algorithm of the chameleon hash, obtains the trapdoor/hash key pair $(y_j, Y_j = y_jP)$.

  u_j's signing key is (x_j, y_j), verification key is (V_j, Y_j).

- Off-Agg-Sig (Done per Aggregation)

  1. Each player u_j chooses at random $(m'_j, r'_j) \in_R \mathcal{M} \times \mathcal{R}$, and computes the chameleon hash value $H'_j = H_{HK}(m'_j, r'_j) = m'_jY_j + r'_jP$.
  2. Each player u_j runs the signing algorithm with private key x_j, output $\sigma'_j = x_j \cdot H'_j$.
  3. Compute the aggregate signature token $\sigma' = \sum_{j=1}^{n} \sigma'_j$. Store the pair (m'_j, r'_j), and the aggregate signature token σ'.

- On-Agg-Sig (Done per Aggregation)

  1. For a given message m_j, player u_j retrieves from memory a random pair (m'_j, r'_j), compute $r_j = (m'_j - m_j)y_j + r'_j \ mod \ q$.
  2. The aggregate signature is $(r_1, \cdots, r_n, \sigma')$.

- On-Agg-Ver (Done per Aggregation)

  On input aggregate signature and verification keys, compute $H'_j = H_{HK}(m_j, r_j) = m_jY_j + r_jP$ and verify the aggregate signature token σ' by checking $e(\sigma', P) = \prod_{j=1}^{n} e(H'_j, V_j)$.

4.2 Our Construction

We construct an efficient on-line/off-line *flexible* aggregate signature scheme $\mathsf{AS}^{\mathsf{On/Off}}$ using Chen *et al*'s key-exposure-free chameleon hash function [3] and BLS short signature [2]. Our construction is composed of the following six algorithms according to the definition in 3.1.

- Sys-Par-Gen (Done Once):

  The algorithm is similar to 4.1, except for the chameleon hash function which is defined as follows:

  $$H_{HK}(m, r) = f(m, K)(K + Y) + rP$$

  Here $f : Z_q \times G \rightarrow Z_q$ is a cryptographic secure hash function. The hash key is $HK = (K, Y)$, where $K = kP, Y = yP$, and $k, y \in_R Z_q^*$.

- On/Off-Agg-Key-Gen (Done Once):

  Input: a set of m players $\mathbb{U} = \{u_1, \cdots, u_m\}$, a security parameter k, the generator P of bilinear group G.

  Common Output: the verification keys, including the chameleon hash and the corresponding signature.

  Private Output: the signing keys for each player u_j.

  1. Each player u_j randomly selects $x_j \in_R Z_q^*$ as his private key, registers $V_j = x_jP$ as his public key.
  2. Each player u_j runs the key generation algorithm of the chameleon hash, obtains the *long-term* hash/trapdoor key pair, denote by $Y_j = y_jP$, $TK = y_j$. u_j randomly selects $m'_j, r'_j, k'_j \in_R Z_q^*$, computes the *one-time* hash key $K'_j = k'_jP$, the chameleon hash $H'_j = H_{HK}(m'_j, r'_j) = f(m'_j, K'_j)(K'_j + Y_j) + r'_jP$, and the corresponding signature $\sigma'_j = x_j \cdot H'_j$. Denote by $\tilde{r_j} = r'_j + f(m'_j, K'_j)(k'_j + y_j)$. Deletes m'_j, r'_j, and k'_j since they are not used anymore.
  3. Player u_j publishes the chameleon hash H'_j and the corresponding signature σ'_j as the *verification keys* of the scheme. Here we use the BLS short signature and σ'_j can be verified by checking whether $e(\sigma'_j, P) = e(H'_j, V_j)$. Note that the chameleon hash function we used here is key exposure free, so the chameleon hash can be *reused*

and the corresponding signature has to be verified only *once*.

The signing key of u_j is $(x_j, y_j, \tilde{r_j})$, and the verification key is (V_j, Y_j, H'_j).

- Off-Agg-Sig (Done per Aggregation):

  Input: the set of n players $\mathcal{U} = \{u_1, \cdots, u_n\} \subseteq \mathbb{U}$, signing key y_j, H'_j, and σ'_j.

  Common Output: the aggregate signature token (σ', H').

  Private Output: *one-time* trapdoor/hash key pairs.

  1. Compute and publish the aggregate signature token $\sigma' = \sum_{j=1}^{n} \sigma'_j$ and $H' = \sum_{j=1}^{n} H'_j$.
  2. Each player u_j chooses at random $k_j \in_R Z_q$, computes $k_j P$ and $k_j + y_j \mod q$, as doing some pre-computation for each signed message m_j given later on-line.
  3. Each player u_j stores the *one-time* trapdoor/hash key pair $(k_j + y_j, k_j P)$.

- Off-Agg-Ver (Done per Aggregation):

  Input: the set of n players $\mathcal{U} = \{u_1, \cdots, u_n\}$, public keys V_j, and the aggregate signature token σ'.

  Common Output: output 1 if the aggregate signature token σ' is valid, 0 otherwise.

  The aggregate signature token σ' of the same player set $\mathcal{U}$ need to be verified only *once* off-line, by checking $e(\sigma', P) = \prod_{j=1}^{n} e(H'_j, V_j)$.

- On-Agg-Sig (Done per Aggregation):

  Input: the set of n players $\mathcal{U} = \{u_1, \cdots, u_n\}$, messages m_j to be signed, trapdoor keys, and pre-computed signature information stored off-line.

  Common Output: trapdoor collision of each H'_j and finally the aggregate signature Σ.

  1. Each player u_j computes $r_j = \tilde{r_j} - f(m_j, K_j) \cdot (k_j + y_j) \mod q$. Note that $H_{HK}(m_j, r_j, K_j) = H'_j$.
  2. Compute $r = \sum_{j=1}^{n} r_j$ and the aggregate signature is

  $$\Sigma = \{K_1, K_2, \cdots, K_n, r, H', \sigma'\}.$$

- On-Agg-Ver (Done per Aggregation):

  Input: the set of n players $\mathcal{U} = \{u_1, \cdots, u_n\}$, the sum of chameleon hash H', hash keys, and the on-line/off-line aggregate signature Σ.

  Output: output 1 if the on-line/off-line aggregate signature Σ is valid, 0 otherwise.

  The verifier checks whether

  $$\sum_{j=1}^{n} H_{HK_j}(m_j, r_j) = rP + \sum_{j=1}^{n} f(m_j, K_j)(K_j + Y_j) \doteq H'.$$

Remark 1. Note that $r_j = \tilde{r_j} - f(m_j, K_j) \cdot (k_j + y_j) \mod q$, so only 1 modular multiplication $f(m_j, K_j) \cdot (k_j + y_j)$ in Z_q is needed in the on-line phase.

Remark 2. Note that by using the *key-exposure-free* chameleon hash function, the hash values and the corresponding signatures can be *reused*, and viewed as verification keys.

4.3 Application in Stock Markets

There are three participants in a stock market, including stockholders, stockbrokers, and stock exchange. One stockbroker serves a great deal of stockholders, and one stock exchange serves many stockbrokers. The trades of stocks from stockholders to stockbroker must be signed.

Consider one stockbroker, who must respond quickly to his stockholders, act as an aggregate party in our scheme; and the stockholders act as aggregate players. The stockholders have much free time to do pre-computation before the opening of the stock exchange. The stockbroker has only one account in the stock exchange, so he aggregates and batch verifies the signatures, then send the aggregate signature to the stock exchange. Our on-line/off-line aggregate signature can be applied to make this process more efficient, as time is money in stock markets.

5 Analysis of the Proposed Scheme

5.1 Security

Theorem 1. *In the aggregate chosen-key model, the proposed on-line/off-line aggregate signature scheme is existentially unforgeable against adaptively chosen message attacks, provided the Discrete Logarithm Problem in G is intractable and Boneh's aggregate signature [1] is existential unforgeable.*

Proof. The proof follows the security argument given by Chen *et al* [3]. Due to the space consideration, we omit it here. □

5.2 Efficiency

We compare the efficiency of two schemes constructed in section 4. We denote by $C(\theta)$ the computation cost of operation θ, and by $|\lambda|$ the bits of λ. Also we denote by $n \cdot BM$ the multiplication of n bilinear map, by M a scalar multiplication in G, by SM a simultaneous scalar multiplication of the form $\lambda P + \mu Q$ in G, and by m the modular multiplication in Z_q. We omit other operations such as point addition and hash in both schemes.

	scheme I	scheme II
off-line phrase	$n \cdot SM + n \cdot M$	$n \cdot M + n \cdot BM$
signing on-line phrase	$n \cdot m$	$n \cdot m$
verifying on-line phrase	$n \cdot SM + n \cdot BM$	$(n+1) \cdot M$

Table 1. Comparison of the computation cost

	scheme I	scheme II
storage off-line phrase	$2n \cdot \lvert q\rvert + 1 \cdot \sigma$	$n \cdot \lvert q\rvert + (n+1) \cdot \lvert l\rvert + 1 \cdot \sigma$
communication on-line phrase	$n \cdot \lvert q\rvert + 1 \cdot \sigma$	$1 \cdot \lvert q\rvert + (n+1) \cdot \lvert l\rvert + 1 \cdot \sigma$

Table 2. Comparison of the storage and communication cost

Since a 160-bit ECC key offers more or less the same level of security as a 1024-bit RSA key, we let $|q| = 160$, and $|l| = 160$. Therefore, our scheme is superior to the key exposure one in the computation cost of the on-line phrase where the storage and communication cost is a little more. So we argue that our scheme is very efficient in signing and verification.

Remark 3. For the same aggregate player set $\mathcal{U}$ in different aggregation rounds, even $n \cdot BM$ can be omitted, since aggregate signature token σ' remains the same. The computation in off-line phrase is much more efficient in this case.

6 Conclusion

In this paper, we introduce the notion of on-line/off-line aggregate signatures and construct a concrete scheme using Chen *et al*'s key-exposure-free chameleon hash function and BLS short signature, based on the "hash-sign-switch" paradigm. To the best of our knowledge, it seems the first on-line/off-line aggregate signature scheme, which is suitable for aggregate applications with "bursty traffic", such as in the stock markets and voting schemes. It can also be used for the environment where the computation and storage resources are limited such as smart card. An interesting challenge is how to reduce the verification information, as we add two random numbers in the chameleon hash function here.

References

[1] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, *Aggregate and Verifiably Encrypted Signatures from Bilinear Maps*, Advances in Cryptology-EUROCRYPT 2003, LNCS 2656, pp. 416-432, Springer-Verlag, 2003.

[2] D. Boneh, H. Shacham, and B. Lynn, *Short Signatures from the Weil Pairing*, Advances In Cryptology-ASIACRYPT 2001, LNCS 2248, pp. 514-532, Springer-Verlag, 2001.

[3] X. Chen, F. Zhang, W. Susilo, and Y. Mu, *Efficient Generic On-line/Off-line Signatures Without Key Exposure*, ACNS 2007, LNCS 4521, pp. 18-30, Springer-Verlag, 2007.

[4] S. Even, O. Goldreich, and S. Micali, *On-line/Off-line Digital Signatures*, Advances in Cryptology-CRYPTO 1989, LNCS 2442, pp. 263-277, Springer-Verlag, 1989.

[5] C. Gentry, and Z. Ramzan, *Identity-Based Aggregate Signatures*, PKC 2006, LNCS 3958, pp. 257-273, Springer-Verlag, 2006.

[6] S. Kent, C. Lynn and K. Seo, *Secure Border Gateway Protocol (secure-bgp)*, IEEE Journal of Selected Areas in Communications, 19(4), pp. 582-592, 2000.

[7] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham, *Sequential Aggregate Signatures from Trapdoor Permutations*, Advances in Cryptology-EUROCRYPT 2004, LNCS 3027, pp. 74-90, Springer-Verlag, 2004.

[8] E. Mykletun, M. Narasimha, and G. Tsudik, *Signature Bouquets: Immutability for Aggregated/Condensed Signatures*, ESORICS 2004, LNCS 3193, pp. 160-176, Springer-Verlag, 2004.

[9] T. Suzuki, Z. Ramzan, H. Fujimoto, C. Gentry, T. Nakayama, and R. Jain, *A System for End-to-End Authentication of Adaptive Multimedia Content*, Communications and Multimedia Security 2004, IFIP 175, pp. 237-249, Springer-Verlag, 2005.

[10] A. Shamir and Y. Tauman, *Improved Online/Offline Signature Schemes*, Advances in Cryptology-CRYPTO 2001, LNCS 2139, pp. 355-367, Springer-Verlag, 2001.